mLoRA: Fine-Tuning LoRA Adapters via Highly-Efficient Pipeline Parallelism in Multiple GPUs

基本信息

under revision in VLDB'25

Zhengmao Ye, Dengchun Li, Zetao Hu, and 7 more authors, IDs Lab, Sichuang University

背景

- 同时微调多个lora adapter具有重要意义
  - 同时训练多种运用于不同领域的LLM
  - 减少LLM训练中确定最优超参数所需时间：同时微调多组具有不同超参数的lora adapter，从而得到最佳的选择。
- 多LoRA Adapter共享base model进行训练可以减少GPU显存占用，同时提高训练并行性。
- 当大量LoRA Adapte所占用的显存超过单个GPU容量时——利用多GPU进行模型并行训练。

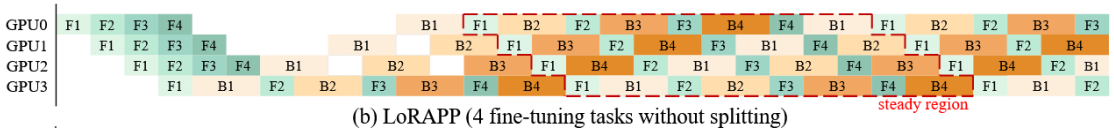
挑战

- **现有的并行方案效率低：**将 base model和adapters分布在多个GPU上。但是现有的并行方法tp、pp，由于需要 GPU 间或计算机间同步，产生高通信开销，且pipeline bubble导致 GPU 利用率低下。
- **GPU内核频繁启动增加训练时间：**并行训练多个小型适配器会导致 GPU 内核频繁启动，增加总训练时间

Observation

- 训练单个LLM时，流水线的各个阶段相互依赖，但是不同lora adapter的训练具有独立特性，能够实现更高的并行度。

  训练一个llm时，各个阶段micro-batch的训练（前向传播和反向传播）应使用相同版本的参数，在1F1B的流水线模式下，会导致同一个micro-batch使用不同版本参数，造成训练效果下降。但在多lora的训练中，不同的lora adapter不存在相互依赖。



(b) LoRAPP (4 fine-tuning tasks without splitting)

  - insight：可以更加自由地为不同的adapter安排不同的训练阶段

- 并行训练多个小型适配器会导致 GPU 内核频繁启动，这会大大增加总训练时间（例如，高达 10%）。训练多个LoRA Adapter的简单做法就是将base model保留在 GPU 上，按顺序切换每个训练任务的Adapter权重。
  - insight：合并不同LoRA Adapter的训练数据，执行集体矩阵乘法运算，好处是GPU内核启动更少。

Algorithm 1 Simply train multiple LoRAs, PyTorch-like.

```
for adapter, data in fine_tuning_task:
    A, B = adapter # swap in the low-rank matrix A and B
    output = data @ W + data @ A @ B
    loss = loss_fn(data, output)
    loss.backward()
```

Algorithm 2 Use the BatchLoRA to train, PyTorch-like.

```
datas = [data for _, data in fine_tuning_task]
adapters = [adapter for adapter, _ in fine_tuning_task]
output = datas @ W # just call once
output += BatchLoRA.apply(datas, adapters)
loss = loss_fn(data, output)
loss.backward()
```

整体架构

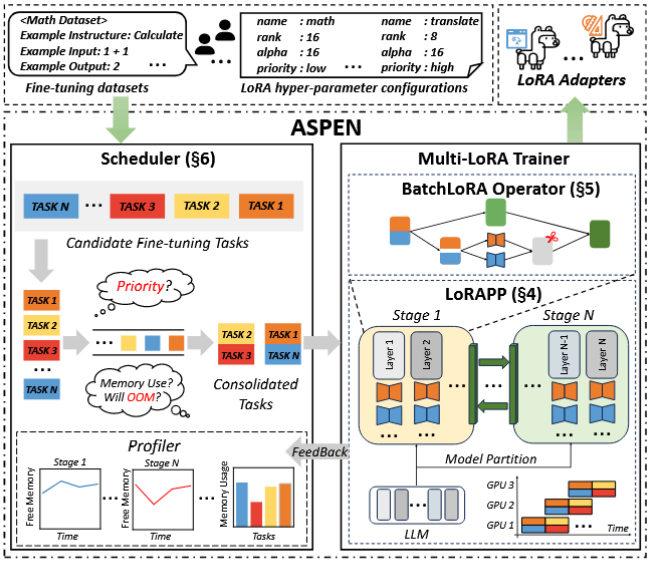


Figure 2: Overview of mLoRA.

设计目标：充分利用多 GPU 资源（包括计算和内存）来实现高微调性能，达到低训练延迟和高训练吞吐量。

架构概述：

- multi-LoRA trainer
  - LoRAPP : LoRA-aware pipeline parallelism
  - BatchLoRA: LoRA-efficient training operator
- task scheduler

LoRAPP

准备阶段

backbone 和lora划分：将base model分到多个GPU中，再将与当前GPU中base model相应线性层关联的那部分适配器分配到相应GPU中实现pp。这里的切分采用均匀的分区方法。

训练阶段

独立性：lora adapter训练不同于llm训练的独立性体现在，每个lora adapter独立累积和应用梯度，不用在不同 LoRA 适配器之间同步梯度。因此允许自由调度不同finetune任务的不同训练阶段（fwd or bwd）。

pipeline并行方式采用1F1B，即最后一个阶段完成fwd之后立马进行bwd，这样做的好处是完成bwd之后可以立即释放中间状态的内存。

将计算和通信重叠，掩盖通信时间

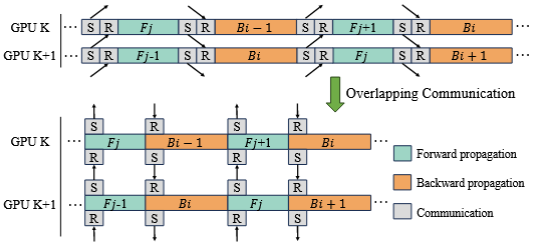


Figure 5: Overlapping communication in LoRAPP. *Fi* **represents the forward propagation of the** *i***th LoRA adapter, while** *Bi* **is its backward propagation.**

LoRAPP的成本分析

bubble ratio

LoRAPP：

$$\text{Bubble Ratio}_{\text{LoRAPP}} = max\{\frac{D-L}{D}, 0\}$$

当L>D时，即可实现0 bubble

GPipe：

$$\text{Bubble Ratio}_{\text{GPipe}} = \frac{D-1}{N+D-1}$$

N的值通常很小，难以实现低bubble ratio

Communication cost

激活值和梯度的总通信量：2（D − 1）Bh

可overlap通信和计算，没有额外的通信开销

Performance analysis

当未达到零气泡状态时，LoRAPP 的吞吐量为 R × L，否则为 R（D + N − 1）/N

其吞吐量可以粗略估计为 R（1 − v）/（1 − μ），其中 R 是 GPipe 的吞吐量，μ 是 GPipe 的气泡比率，v 是 LoRAPP 的气泡比率

Memory usage

LoRAPP节省(L − 1)W0显存

BatchLoRA

即使采用多lora的流水线并行，GPU的利用率和内存利用率仍然很低，lora的训练任务较为简单，无法充分利用GPU的处理能力。

在内存限制条件下增加并行的lora个数，然而并行训练多个小型适配器会导致 GPU 内核频繁启动，这会大大增加总训练时间

因此提出BatchLoRA 将每个GPU上的多个lora合并起来进行前向和后向传播，减少内核启动开销

前向传播和反向传播

$$H = \begin{pmatrix} h_1 \\ \vdots \\ h_n \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} W + \begin{pmatrix} x_1 A_1 B_1 \\ \vdots \\ x_n A_n B_n \end{pmatrix} = XW + \begin{pmatrix} x_1 A_1 B_1 \\ \vdots \\ x_n A_n B_n \end{pmatrix} \quad (2)$$

- backbone部分合并计算$Y = XW$

$$\begin{pmatrix} \nabla A_1 \\ \vdots \\ \nabla A_n \end{pmatrix} = \begin{pmatrix} x_1{}^\intercal \nabla h_1 B_1{}^\intercal \\ \vdots \\ x_n{}^\intercal \nabla h_n B_n{}^\intercal \end{pmatrix}, \begin{pmatrix} \nabla B_1 \\ \vdots \\ \nabla B_n \end{pmatrix} = \begin{pmatrix} A_1{}^\intercal x_1{}^\intercal \nabla h_1 \\ \vdots \\ A_n{}^\intercal x_n{}^\intercal \nabla h_n \end{pmatrix} \quad (3)$$

$$\nabla X = \begin{pmatrix} \nabla x_1 \\ \vdots \\ \nabla x_n \end{pmatrix} = \nabla H W^\intercal + \begin{pmatrix} \nabla h_1 B_1{}^\intercal A_1{}^\intercal \\ \vdots \\ \nabla h_n B_n{}^\intercal A_n{}^\intercal \end{pmatrix}, \nabla H = \begin{pmatrix} \nabla h_1 \\ \vdots \\ \nabla h_n \end{pmatrix} \quad (4)$$

- $\nabla H W^T$合并计算

对于这两部分，各自只需启动一次GPU内核进行计算，减少内核启动带来的开销。不用为每个 LoRA Adapter启动矩阵乘法运算 $x_i W$和 $\nabla h_i W^T$。

计算图剪枝：各个adapter B A的梯度分开计算，再将各自计算得到的∇xi，直接加到∇Y上

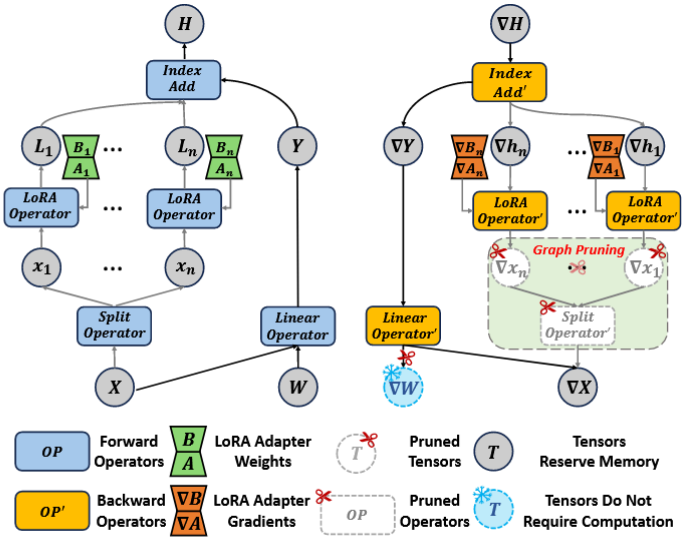
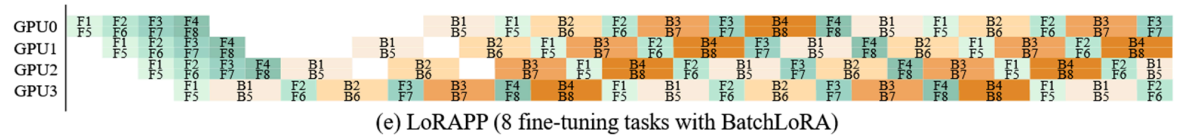


Figure 6: Computational graphs of BatchLoRA operator with graph pruning.

BatchLoRA和LoRAPP相结合



(e) LoRAPP (8 fine-tuning tasks with BatchLoRA)

- 尽可能将微调任务的数量与 GPU 的数量相匹配来实现 zero bubble 。
- BatchLoRA 整合多个LoRA finetune任务，在 GPU 内存允许的范围内安排尽可能多的任务进行组合，同时确保组合任务的数量等于 GPU 的数量，以保持这种zero bubble状态。

成本分析

内核启动成本分析

不使用 BatchLoRA 运算符微调 k 个 LoRA 适配器时：内核启动成本为 kα + kβ。

使用 BatchLoRA 算子时，预训练模型使用合并数据进行一次完整的正向和向后传播，每个 LoRA 适配器使用训练数据进行一次完整的正向和反向传播：内核启动成本为 α + kβ

由于β ≪ α，内核启动成本的降低约为 （k −1）/k，k为同时训练的 LoRA 适配器的数量

batchlora操作cost分析

减少峰值内存占用

Task scheduler

调度目标：尽可能多调度finetune任务，同时满足用户优先级、避免oom

策略：分配优先级，按照优先级顺序调度，优先级相同的情况按照fcfs

调度时机：在每次Iteration结束时进行调度决策

内存估计模型：估计每个finetune任务的所需内存，根据输入的请求长度、bsz采用非线性最小二乘求解器进行拟合：

$$Mem = \beta_0 + \beta_1 B_t L_n + \beta_2 B_t L_n^2 \qquad (5)$$

Ln 是输入训练数据序列长度;Bt 是输入批量大小；β0、β1 和 β2 是非负系数。

实验

实验设置

models：Llama2-13B 、Llama2-7B 、TinyLlama-1.1B

platforms：

- 单机 单GPU模式 A6000
- 单机 多GPU模式 4×A6000 48GB 通过PCIe 4.0x16连接
- 多机 多GPU模式 8×3090 24GB 通过 1Gbps 网络连接

workload：

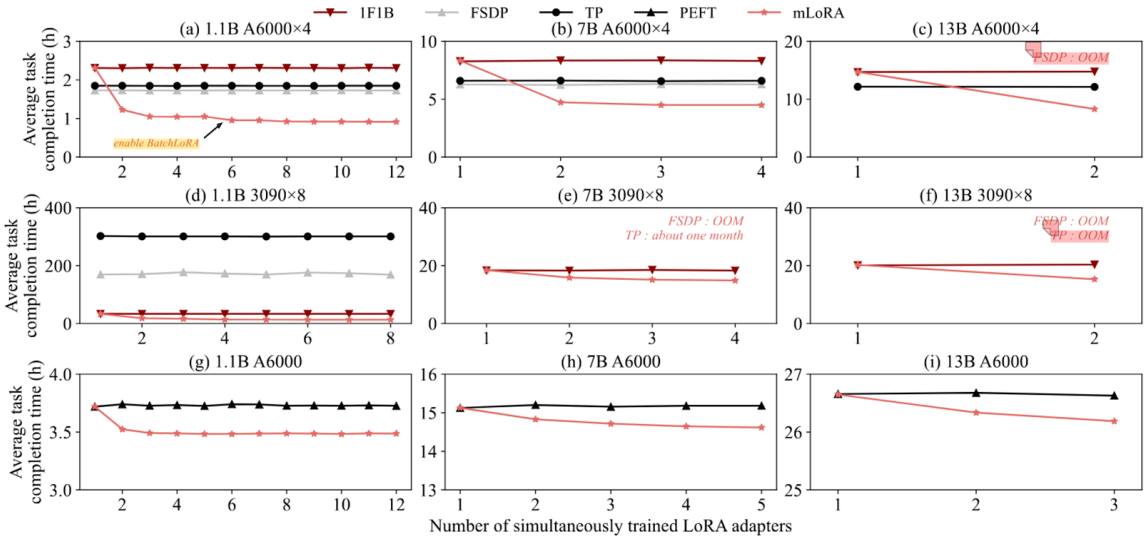
- 采用数据集GSM8K进行训练
- bsz=8，seqlen=512，epoch=10，r=16
- lora adapter运用在k、q、v、o参数矩阵上

Metrics：平均任务完成时间、系统吞吐量

baseline：

- 单GPU环境：PEFT
- 多GPU环境：1F1B、TP、FSDP

端到端性能测试



- 单机多GPU：比起TP、FSDP减少了通信时间，FSDP由于需要额外的参数复制无法训练更大的模型。
- 多机多GPU：FSDP 和 TP 在每个节点上都会产生额外的内存开销，1F1Bbubble ratio较LoRAPP高。此外，通信是在该设置下的瓶颈，因此 7B 和 13B 模型的训练时间几乎相同
- 单机单GPU：将平均任务完成时间缩短了 8%，减少的这部分就是启动内核函数的开销。随着基本模型大小的增加，启动内核函数的开销所占的比例越来越小，导致 BatchLoRA 的性能提升降低（例如，13B 模型的性能提升减少了 2%）

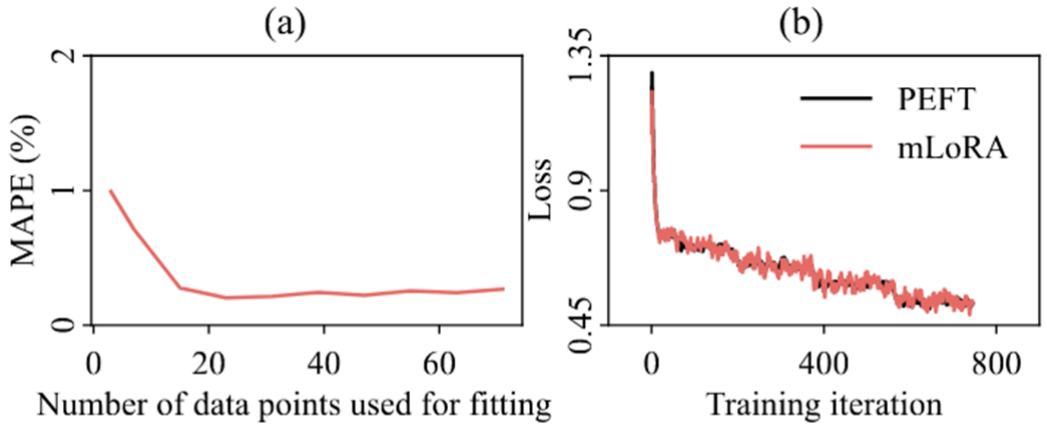


Figure 8: (a) The accuracy of the online model fitting. (b) Model training convergence study.

- 即使只有少量的数据点也能使得内存估计模型达到0.25%的MAPE精度。
- mLoRA 表现出与 PEFT 相似的收敛趋势，表明 mLoRA 实现了与 PEFT 相同的性能.

LoRAPP并行策略的有效性

单机多GPU实验环境

- 分析了LoRAPP与1F1B方法在bubble比率上的性能差异。

  影响因素：microbatch的数量、adapter的数量

  bubble ratio比1F1B要低
- 比较了LoRAPP与TP方法在通信量上的差异。

  LoRAPP 和 1F1B 的通信量相同，但明显小于 TP。
- 吞吐量：在不同数量的GPU上训练LoRA模型，评估mLoRA的吞吐量，并与1F1B、TP和FSDP进行比较。
- 扩展性：测试了LoRAPP的线性可扩展性，通过在2至8个GPU上训练模型，mLoRA 的吞吐量随 GPU 数量的增加而线性增加。